**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ**

**Навчально-науковий інститут автоматики кібернетики та обчислювальної техніки**

КУРСОВА РОБОТА

на тему:

“Створення веб-браузерної гри судоку, з генерація ігрового поля”

Виконав:

студент групи: ІПЗ-21

Струк Андрій Володимирович

Керівник:

к.т.н, доцент,

Жуковський Віктор Володимирович

Рівне – 2022

# ЗМІСТ

**[ВСТУП](#_uvl6nl6bz0cg) 3**

[Мета та завдання проекту](#_hzbxpa8d8jzo) 3

**[РОЗДІЛ 1. ЖИТТЄВИЙ ЦИКЛ ПРОЕКТА](#_ke1hd63bdxcm) 3**

[1.1 Аналіз](#_latcbjbsh9sr) 3

[1.1.1 Ідея Розробки](#_psdjdb5buo05) 3

[1.1.2 Функціонал гри та постановка вимог](#_6c8impb2zt2t) 3

[1.2 Проектування](#_g2bx8aaq4p7w) 4

[1.2.1. Вибір середовища реалізації та ресурсів](#_8zdnmkx2a6ea) 4

[1.2.2. Створення HTML сторінки](#_uo6ybuq1hzfw) 4

[1.2.3. Створення алгоритму генерації ігрового поля](#_xwa1h1jq5i89) 4

[1.2.4.](#_abn438uiakvm) Відображення Судоку на веб-сторінці 14

[1.2.5.](#_m9jhs2cwbyc7) Створення дизайну сторінки 15

**[ВИСНОВКИ](#_688yg0xlknm8) 30**

**[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ](#_myvj9ihjajkj) 31**

# 

## Мета та завдання проекту

1. Організація роботи з розробки стратегії ігрового проекту
2. Розробка правил гри і умов
3. Створення генеративної структури проекта
4. Вибрати технології для реалізації ігрового проекта
5. Підведення підсумків ігрового проекту

# РОЗДІЛ 1. ЖИТТЄВИЙ ЦИКЛ ПРОЕКТА

## 1.1 Аналіз

### 1.1.1 Ідея Розробки

* Ідеєю є створення гри судоку в веб-додатку з використанням алгоритму генерації ігрового поля. Створення алгоритму розв’язання головоломки. Також виділення однакових чисел, головних секторів, стовпців і рядків та помилок гравця.

1.1.2 Аналіз гри “Судоку”

Судоку - це головоломка, яка полягає в розміщенні цифр в таблиці розміром 9x9 так, щоб кожна цифра від 1 до 9 зустрічалася в кожному рядку, кожному стовпчику і в кожному блоку 3x3 без повторень. Ось основні правила гри в судоку:

* Розміщення цифр: Головна мета гри - заповнити всю таблицю 9x9 цифрами від 1 до 9, дотримуючись правил.
* Рядки: Кожен рядок повинен містити всі цифри від 1 до 9, і жодна цифра не повинна повторюватися в одному рядку.
* Стовпчики: Кожен стовпчик також повинен містити всі цифри від 1 до 9, і жодна цифра не повинна повторюватися в одному стовпчику.
* Блоки 3x3: Таблиця поділена на 9 блоків 3x3. Кожен блок також повинен містити всі цифри від 1 до 9, і жодна цифра не повинна повторюватися в одному блоку.
* Початковий стан: Головоломка розпочинається з певними цифрами, які вже заповнені в таблиці. Ці цифри служать вам початковими вказівками і визначаються таким чином, щоб гарантувати єдиний вірний шлях до розв'язку.
* Логічні висновки: Для розв'язання судоку використовуйте логічні висновки і виключення. Розглядаючи рядки, стовпчики та блоки, додавайте цифри так, щоб вони відповідали умовам і не порушували правил гри.
* Cпроби та помилки: Розв'язування судоку може вимагати декількох спроб та помилок. Перевіряйте різні комбінації цифр, доки не знайдете правильну послідовність.
* Терпіння та практика: Гра в судоку вимагає терпіння і практики. Чим більше ви граєте, тим краще розвивається ваша здатність робити логічні висновки і знаходити правильні рішення.

Таким чином ми провели аналіз самої гри Судоку

### 1.1.3 Функціонал гри та постановка вимог

Виділимо основні частини, які повинні бути присутні в грі:

Створення ігрової сесії :

* Генерація ігрового роля
* Виділення головних секторів, стовпців і рядків
* Виділення помилок гравця та виділення однакових чисел

Ігрова сесія:

* Починається після створення ігрового поля і закінчується при вирішені головоломки.
* У гравця є необмежена кількість часу та похибок.
* Для зручності користувача будуть додані властивості комірок які даватимуть можливість підсвічування чисел, суміжних рядків та стовпців

## 

## 1.2 Проектування

### 1.2.1. Вибір середовища реалізації та ресурсів

* Редактор коду**:***Visual Studio Code.*
* Мова програмування: *Java script.*
* Мова розмітки: *HTML.*
* Мова стилів: *CSS.*

### 1.2.2 Створення HTML сторінки

<!DOCTYPE html>

<html lang = "uk">

    <head>

        <meta charset="UTF-8">

        <meta mane="viewport" content="width=device-width innitial-scale=1.0 user-scalable=no">

        <title>Sudoku</title>

        <link rel="stylesheet" href="sudoku.css">

    </head>

    <body>

        <h1>Sudoku</h1>

        <hr>

        <div id="app"></div>

        <script src="Sudoku.js"></script>

        <script src="script.js"></script>

    </body>

</html>

### 1.2.3. Створення алгоритму генерації ігрового поля

Створення Конструктора класу ‘Sudoku’.

1. Конструктор приймає параметр initString, який представляє рядок з початковим станом судоку. За замовчуванням, він має рядок з 81 нулем, що відповідає порожньому судоку.
2. Конструктор розбиває initString на масив чисел та ініціалізує внутрішнє поле body, яке представляє собою 81 комірку судоку.

3. Кожна комірка має ряд властивостей, таких як id, x, y, number, selected, supported, important, error, started, та s. Ці властивості використовуються для відстеження стану та взаємодії з комірками судоку.

class Sudoku

{

    constructor (initString = '000000000000000000000000000000000000000000000000000000000000000000000000000000000')

    {

        const startValues = initString

            .split("")

            .filter(x => "0123456789".includes(x))

            .map(x => Number(x))

        this.body = []

        let idCounter = 1

        for (let y = 0; y < 9; y++)

        {

            for (let x = 0; x < 9; x++)

            {

                this.body.push({

                    id: idCounter,

                    x,

                    y,

                    number: startValues[idCounter - 1],

                    selected: false,

                    supported: false,

                    important: false,

                    error: false,

                    started: startValues[idCounter - 1] === 0 ? false : true,

                    s: parseInt(y / 3) \* 3 + parseInt(x / 3)

Вираз s: parseInt(y / 3) \* 3 + parseInt(x / 3) використовується для визначення "сегмента" (або "блоку") судоку, до якого належить певна клітинка

parseInt(y / 3): Це обчислює номер рядка, в якому знаходиться клітинка, і ділить його на 3. Результатом буде ціле число, яке вказує на те, в якому рядку судоку знаходиться дана клітинка, поділене на 3.

Аналогічно і вираз parseInt(x / 3) обчислює номер стовпця, в якому знаходиться клітинка, і ділить його на 3. Результат також буде цілим числом і вказує на номер стовпця, поділений на 3.

Після цього результати обох обчислень додаються разом і множаться на 3: (parseInt(y / 3) \* 3 + parseInt(x / 3)). Це об'єднує інформацію про номер рядка та номер стовпця, поділену на 3, і повертає номер сегмента, до якого належить клітинка. Результатом є число від 0 до 8, яке ідентифікує сегмент.

                })

                idCounter++

            }

        }

}

Статичні методи getFreeCell та getCloseCell:

getFreeCell(sudoku): Повертає випадкову порожню комірку з переданого об'єкта sudoku.

getCloseCell(sudoku): Повертає випадкову заповнену комірку з переданого об'єкта sudoku.

static getFreeCell (sudoku)

    {

        const cells = sudoku.body.filter(x => !x.number)

        const index = Math.floor(Math.random() \* cells.length)

        return cells[index]

    }

    static getCloseCell (sudoku)

    {

        const cells = sudoku.body.filter(x => x.number)

        const index = Math.floor(Math.random() \* cells.length)

        return cells[index]

}

Статичний метод generate:

generate(n): Генерує новий екземпляр гри судоку з n заповненими комірками.

Спочатку заповнює всі комірки випадковими числами.

Затим вирішує судоку з такими числами і потім видаляє деякі числа, щоб залишити лише n заповнених комірок.

    static generate(n)

    {

        n = Math.min(81, Math.max(n, 0))

        const w = (new Sudoku)

        for(let i = 0; i <= 9; i++)

        {

            const freeCell = Sudoku.getFreeCell(w)

            freeCell.number = i

        }

        const s = w.solve()

        for (let i = 0; i < 81 - n; i++)

        {

            const closeCell = Sudoku.getCloseCell(s)

            closeCell.number = 0

        }

        return new Sudoku(s.body.map(x => x.number).join(''))

    }

Метод isSolved:

isSolved: Перевіряє, чи головоломка судоку вирішена. Перевіряє, чи в кожному ряду, кожному стовпці та в кожному сегменті є всі числа від 1 до 9 без повторень.

    get isSolved ()

    {

        for(const cell of this.body)

        {

            if (cell.number === 0)

            {

                return false

            }

        }

        for(let i = 0; i < 9; i++)

        {

            const row = this.getRow(i).map(x => x.number)

            for(let n = 1; n <= 9; n++)

            {

                if(!row.includes(n))

                {

                    return false

                }

            }

            const column = this.getColumn(i).map(x => x.number)

            for(let n = 1; n <= 9; n++)

            {

                if(!column.includes(n))

                {

                    return false

                }

            }

            const segment = this.getSegment(i).map(x => x.number)

            for(let n = 1; n <= 9; n++)

            {

                if(!segment.includes(n))

                {

                    return false

                }

            }

        }

        return true

}

Методи getCopy, getRow, getColumn та getSegment:

getCopy(): Повертає копію поточного об'єкта Sudoku.

getRow(n): Повертає рядок n.

getColumn(n): Повертає стовпець n.

getSegment(n): Повертає сегмент n.

     getCopy ()

    {

        return new Sudoku(this.body.map(x => x.number).join(""))

    }

    getRow (n)

    {

        const row = []

        for(let i = 0; i < 9; i++)

        {

            row.push(this.body[9 \* n + i])

        }

        return row;

    }

    getColumn (n)

    {

        const column = []

        for(let i = 0; i < 9; i++)

        {

            column.push(this.body[9 \* i + n])

        }

        return column

    }

    getSegment (n)

    {

        const segment = []

        const x = n % 3

        const y = parseInt(n / 3)

        for(let dy  = 0; dy <3; dy++)

        {

            for(let dx  = 0; dx <3; dx++)

            {

                segment.push(this.body[ y \* 27 + dy \* 9 + x \* 3 +dx])

            }

        }

        return segment

}

Методи keydownHandler, focusHandler та blurHandler:

keydownHandler(event, cell): Обробляє події натискання клавіш для введення чисел та взаємодії з комірками.

focusHandler(event, cell): Обробляє події отримання фокусу на комірку.

blurHandler(event, cell): Обробляє події втрати фокусу на комірку.

    keydownHandler(event, cell)

    {

        if(!cell.started)

        {

            if("123456789".includes(event.key))

            {

                cell.number = parseInt(event.key)

                if(cell.error)

                {

                    for (const item of this.body)

                    {

                        item.error = false

                    }

                }

                for(const item of this.getRow(cell.y))

                {

                    if(item === cell)

                    {

                        continue

                    }

                    if(item.number === cell.number)

                    {

                        item.error = true

                        cell.error = true

                    }

                }

                for(const item of this.getColumn(cell.x))

                {

                    if(item === cell)

                    {

                        continue

                    }

                    if(item.number === cell.number)

                    {

                        item.error = true

                        cell.error = true

                    }

                }

                for(const item of this.getSegment(cell.s))

                {

                    if(item === cell)

                    {

                        continue

                    }

                    if(item.number === cell.number)

                    {

                        item.error = true

                        cell.error = true

                    }

                }

            }

            else if(["Backspace", "Delete"].includes(event.key))

            {

                cell.number = 0

            }

            for(const item of this.body)

            {

                item.important = false

            }

            if(cell.number)

            {

                for(const item of this.body)

                {

                    if(item.number === cell.number)

                    item.important = true

                }

            }

        }

        event.preventDefault()

        this.viewUpdate()

    }

    focusHandler(event, cell)

    {

        cell.selected = true

        for (const item of this.getRow(cell.y))

        {

            item.supported = true

        }

        for (const item of this.getColumn(cell.x))

        {

            item.supported = true

        }

        if(cell.number)

        {

            for(const item of this.body)

            {

                if(item.number === cell.number)

                item.important = true

            }

        }

        this.viewUpdate()

    }

    blurHandler(event, cell)

    {

        cell.selected = false

        if(cell.error)

        {

            cell.number = 0

        }

        for(const cell of this.body)

        {

            cell.error = false

            cell.important = false

            cell.supported = false

        }

        this.viewUpdate()

}

Метод getHTML та viewUpdate:

getHTML(size): Генерує HTML-представлення гри судоку з заданою шириною size.

Метод viewUpdate() в класі Sudoku відповідає за оновлення відображення гри на веб-сторінці. Цей метод проглядає всі клітинки гри і встановлює їхнім HTML-елементам класи та значення відповідно до стану кожної клітинки.

1. for (const cell of this.body): Це початок циклу, який перебирає всі клітинки гри, які зберігаються у масиві this.body.
2. cell.element.classList.remove(...): В цьому рядку здійснюється видалення всіх класів, які можуть бути застосовані до HTML-елементу клітинки. Це робиться за допомогою методу classList.remove(), і з нього видаляються всі класи, які можуть бути встановлені для цієї клітинки.
3. cell.element.value = cell.number ? cell.number : "": Ця лінія встановлює значення для HTML-елементу клітинки. Якщо cell.number (номер клітинки) не дорівнює нулю, то встановлюється сам номер. В іншому випадку, якщо номер рівний 0 (клітинка порожня), то встановлюється порожній рядок.
4. Подальші умовні перевірки (if...) встановлюють класи для HTML-елементу клітинки в залежності від її стану.

if (cell.supported): Якщо cell.supported відповідає true, то додається клас "supported-cell" до HTML-елементу.

if (cell.selected): Якщо cell.selected відповідає true, то додається клас "selected-cell".

if (cell.important): Якщо cell.important відповідає true, то додається клас "important-cell".

if (cell.error): Якщо cell.error відповідає true, то додається клас "error-cell".

Отже, метод viewUpdate() встановлює класи та значення для HTML-елементів клітинок, щоб відображати їх стан на веб-сторінці. Це дозволяє користувачеві бачити розташування та стан клітинок у грі.

    getHTML (size)

    {

        for (const cell of this.body)

        {

            const inputElement = document .createElement("input")

            inputElement.classList.add("sudoku-cell")

            inputElement.setAttribute("type", "text")

            inputElement.addEventListener("keydown", event => this.keydownHandler(event, cell))

            inputElement.addEventListener("focus", event => this.focusHandler(event, cell))

            inputElement.addEventListener("blur", event => this.blurHandler(event, cell))

            if(cell.started)

            {

                inputElement.classList.add("start-cell")

            }

            cell.element = inputElement

        }

        const rootElement = document.createElement("div")

        rootElement.classList.add("sudoku-game")

        rootElement.style.width = `${size}px`

        rootElement.style.height = `${size}px`

        rootElement.style["font-size"] = `${size / 20}px`

        rootElement.style.position =`center`

        for(let s = 0; s < 9; s++)

        {

            const segmentElement = document.createElement("div")

            segmentElement.classList.add("sudoku-segment")

            for (const cell of this.getSegment(s))

            {

                segmentElement.append(cell.element)

            }

            rootElement.appendChild(segmentElement)

        }

        this.viewUpdate()

        return rootElement

    }

    viewUpdate()

    {

        for (const cell of this.body)

        {

            cell.element.classList.remove("error-cell","important-cell", "supported-cell", "selected-cell")

            cell.element.value = cell.number ? cell.number : ""

            if(cell.supported)

            {

                cell.element.classList.add("supported-cell")

            }

            if(cell.selected)

            {

                cell.element.classList.add("selected-cell")

            }

            if(cell.important)

            {

                cell.element.classList.add("important-cell")

            }

            if(cell.error)

            {

                cell.element.classList.add("error-cell")

            }

        }

}

Метод getPotentials() у класі Sudoku використовується для обчислення потенційних варіантів для кожної порожньої клітини у грі. При обчисленні варіантів для кожної клітини враховуються вже заповнені клітини в тому ж рядку, стовпці та сегменті судоку.

1. Спочатку створюється порожній масив potentials, який буде містити потенційні варіанти для кожної клітини.
2. Потім виконується цикл for...of, який перебирає всі клітини у this.body.
3. Внутрішня умова перевіряє, чи вже є число (cell.number) у даній клітині. Якщо так, то це число додається до потенціалів для цієї клітини.
4. Якщо клітинка порожня (тобто cell.number дорівнює 0), то обчислюються потенційні варіанти для неї. Для цього здійснюється наступне:

* З клітини отримуються номери вже заповнених клітин в тому ж рядку, використовуючи метод getRow(cell.y).map(x => x.number).
* З клітини отримуються номери вже заповнених клітин у тому ж стовпці, використовуючи метод getColumn(cell.x).map(x => x.number).
* З клітини отримуються номери вже заповнених клітин у тому ж сегменті, використовуючи метод getSegment(cell.s).map(x => x.number).

1. Створюється масив alphabet з числами від 1 до 9, які представляють всі можливі числа в судоку.
2. Використовуючи фільтрацію масивів, обчислюються потенційні варіанти для цієї клітини:

* Варіанти, які відсутні в rowNumbers, фільтруються за допомогою .filter(x => !rowNumbers.includes(x)).
* З фільтрованих варіантів видаляються ті, які вже є в columnNumbers і segmentNumbers, використовуючи подібні фільтри.
* Результатом є масив, який містить потенційні варіанти для даної порожньої клітини.

1. Отримані потенціальні варіанти додаються до масиву potentials для даної клітини.
2. Цей процес повторюється для кожної клітини у грі.

На завершення, масив potentials містить потенційні варіанти для кожної клітини, і він повертається з методу.

    getPotentials()

    {

        const potentials = []

        for(const cell of this.body)

        {

            if(cell.number)

            {

                potentials.push(cell.number)

            }

            else

            {

                const rowNumbers = this.getRow(cell.y).map(x => x.number)

                const columnNumbers = this.getColumn(cell.x).map(x => x.number)

                const segmentNumbers = this.getSegment(cell.s).map(x =>x.number)

                const alphabet = [1, 2, 3, 4, 5, 6, 7, 8, 9]

                potentials.push(

                    alphabet

                    .filter(x => !rowNumbers.includes(x))

                    .filter(x => !columnNumbers.includes(x))

                    .filter(x => !segmentNumbers.includes(x))

                )

            }

        }

        return potentials

Метод solve() в класі Sudoku відповідає за спробу розв'язання гри в судоку. Він використовує рекурсивний алгоритм для визначення можливих рішень гри.

1. Спочатку створюється копія поточної гри, використовуючи метод this.getCopy(). Ця копія буде використовуватися для спроб розв'язати судоку, не впливаючи на поточну гру.
2. Встановлюється початковий стан флагу flagsolve на true. Цей флаг використовується для слідкування за тим, чи вдалося вирішити судоку під час поточної ітерації.
3. Запускається цикл while (flagsolve), який виконується доти, доки флаг flagsolve залишається true. Цей цикл виконується для спроби розв'язати гру, використовуючи прості методи.
4. Усередині цього циклу отримуються потенційні варіанти для всіх клітин судоку за допомогою const potentials = copy.getPotentials().
5. Далі, у циклі for(let i = 0; i < 81; i++) перевіряється кожна клітинка судоку. Якщо у клітини є лише один потенційний варіант (тобто potential є масивом і має довжину 1), то цей варіант встановлюється як номер для клітини, і флаг flagsolve встановлюється в true, щоб позначити, що в цій ітерації знайдено розв'язок для щонайменше однієї клітини.
6. Після виконання першого циклу перевіряються більш складні сценарії. Зовнішній цикл for(let power = 2; power <= 9; power++) перебирає можливі потужності (кількість потенційних варіантів) від 2 до 9.
7. Усередині цього зовнішнього циклу, внутрішній цикл for(let i = 0; i < 81; i++) перебирає всі клітини судоку.
8. Якщо для клітини існує power різних потенційних варіантів (тобто potentials[i].length === power), то виконується внутрішній цикл, який спробує кожен з цих потенційних варіантів.
9. Для кожного потенційного варіанту створюється копія гри nextCopy і встановлюється вибраний варіант як номер клітини.
10. За допомогою рекурсії, викликається метод solve() для nextCopy для спроби розв'язати гру з встановленим варіантом.
11. Якщо рекурсивний виклик повертає гру, яка вирішена (тобто resultCopy.isSolved дорівнює true), то ця гра повертається як результат.
12. Якщо розв'язок не знайдено, цей процес повторюється для інших клітин і потужностей, доки не буде знайдено розв'язок або не вичерпаються всі можливі комбінації.
13. На завершення, якщо жоден з ітераційних циклів не призвів до розв'язку, метод повертає початкову гру copy.

    }

    solve ()

    {

        const copy = this.getCopy()

        let flagsolve = true

        while (flagsolve)

        {

            flagsolve = false

            const potentials = copy.getPotentials()

            for(let i = 0; i < 81; i++)

            {

                const potential = potentials[i]

                if(potential instanceof Array && potential.length === 1)

                {

                    copy.body[i].number = potential[0]

                    flagsolve = true

                }

            }

        }

        const potentials = copy.getPotentials()

        mainLoop:

        for(let power = 2; power <= 9; power++)

        {

            for(let i = 0; i < 81; i++)

            {

                if(potentials[i].length === power)

                {

                    for(const value of potentials[i])

                    {

                        const nextCopy = copy.getCopy()

                        nextCopy.body[i].number = value

                        const resultCopy = nextCopy.solve()

                        if(resultCopy.isSolved)

                        {

                            return resultCopy

                        }

                    }

                    break mainLoop

                }

            }

        }

        return copy

    }

}

1.2.4 Відображення Судоку на веб-сторінці

Цей код генерує головоломку судоку, а потім відображає її на веб-сторінці. Давайте розглянемо кожний рядок коду:

const field = Sudoku.generate(50): Спочатку створюється головоломка судоку за допомогою статичного методу generate класу Sudoku. Генерується головоломка з 50 заповненими комірками. Отриманий об'єкт field містить головоломку.

document.querySelector('#app'): Ця строка використовує document.querySelector для пошуку елемента з ідентифікатором "app" в DOM (Document Object Model).

.append(field.getHTML(750)): Метод .append() вставляє HTML-представлення головоломки судоку в знайдений елемент з ідентифікатором "app". field.getHTML(750) викликає метод getHTML() головоломки та передає розмір 750 для створення відображення головоломки з відповідним розміром.

Отже, цей код генерує головоломку судоку та відображає її на сторінці в елементі з ідентифікатором "app".

const field = Sudoku.generate(50)

    document

    .querySelector('#app')

    .append(field.getHTML(750))

1.2.5 Створення дизайну сторінки

body

{

    font-family: sans-serif;

    text-align: center;

}

hr

{

    width: 500px;

}

#buttons

{

    display: flex;

    flex-wrap: wrap;

    justify-content: center;

}

#app

{

    display: flex;

    flex-wrap: wrap;

    justify-content: center;

}

.sudoku-game

{

    display: flex;

    flex-wrap: wrap;

}

.sudoku-segment

{

    border: 1px solid darkgray;

    margin: 1px;

    padding: 1px;

    width: calc(100%/3 - 6px);

    height: calc(100%/3 - 6px);

    display: flex;

    flex-wrap: wrap;

    align-content: flex-start;

}

.sudoku-cell

{

    text-align: center;

    font-size: 100%;

    width: calc(100%/3 - 2px);

    height: calc(100%/3 - 2px);

    border:  none;

    outline:  none;

    padding: 0px;

    margin: 1px;

    background-color:  #CCCCCC;

}

.sudoku-cell.start-cell

{

    color: #333333 ;

}

.sudoku-cell.important-cell

{

    color: #1faf7f;

}

.sudoku-cell.error-cell

{

    color:red;

}

.sudoku-cell.supported-cell

{

    background: #ECECEC;

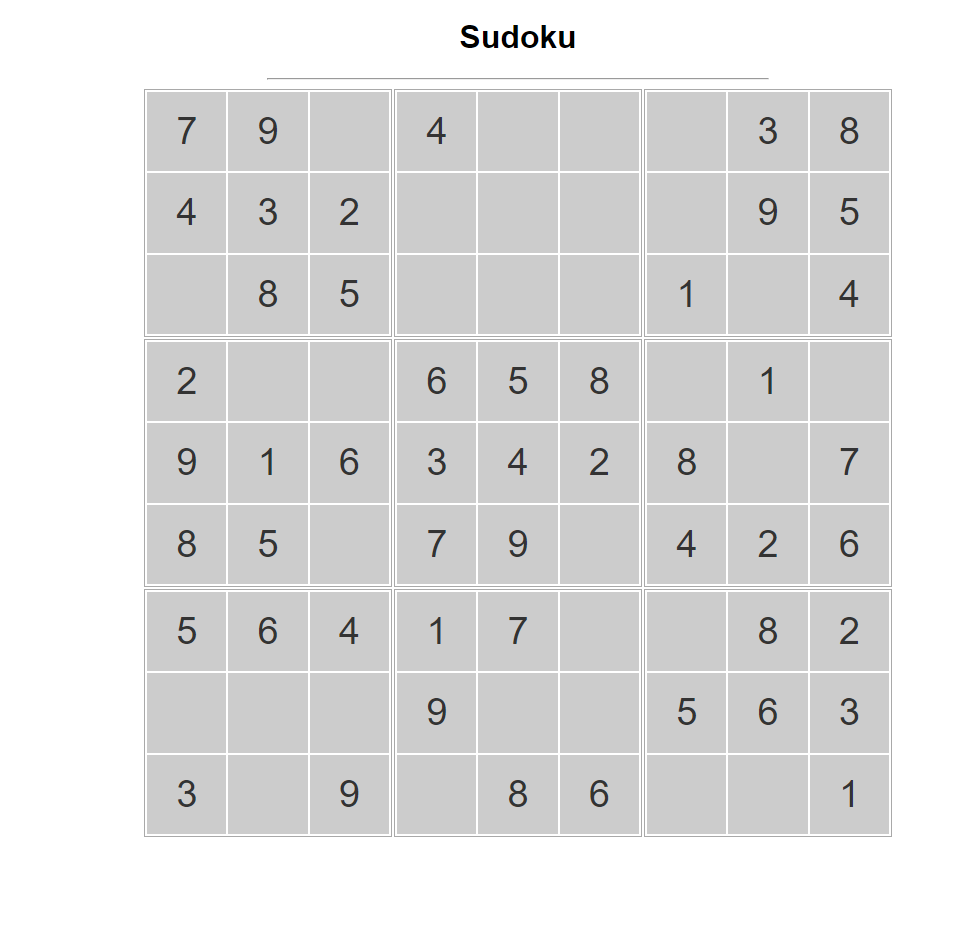
}

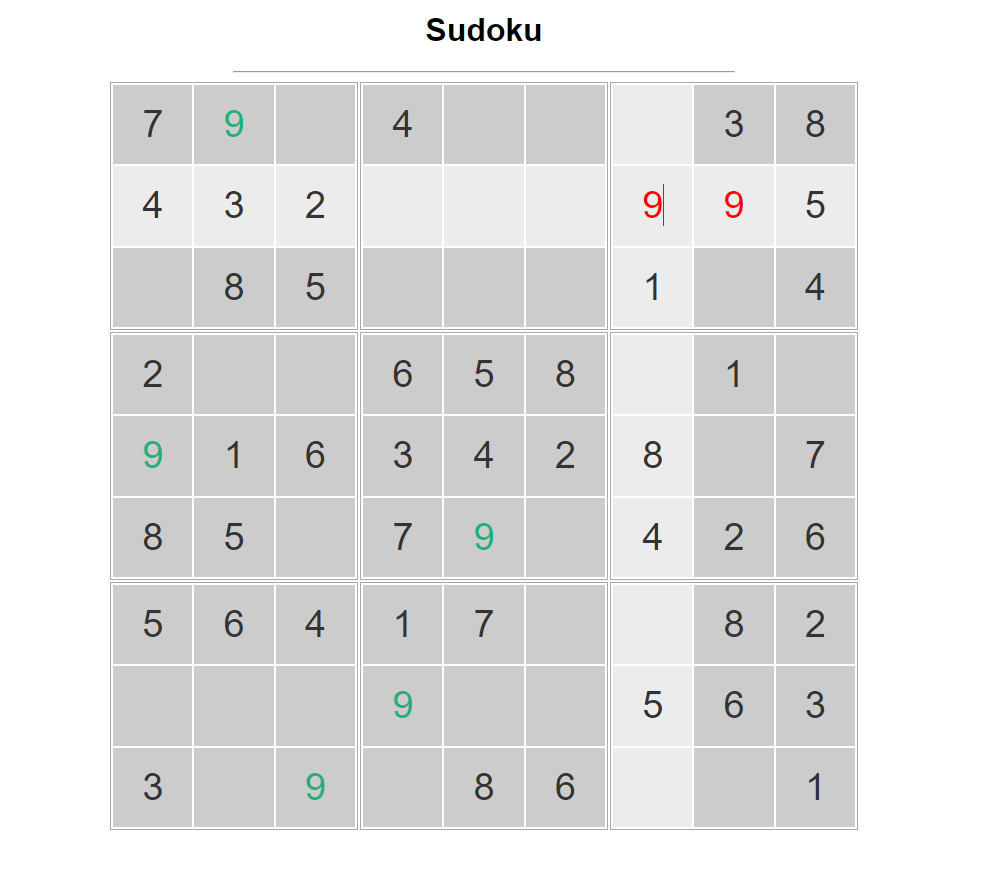
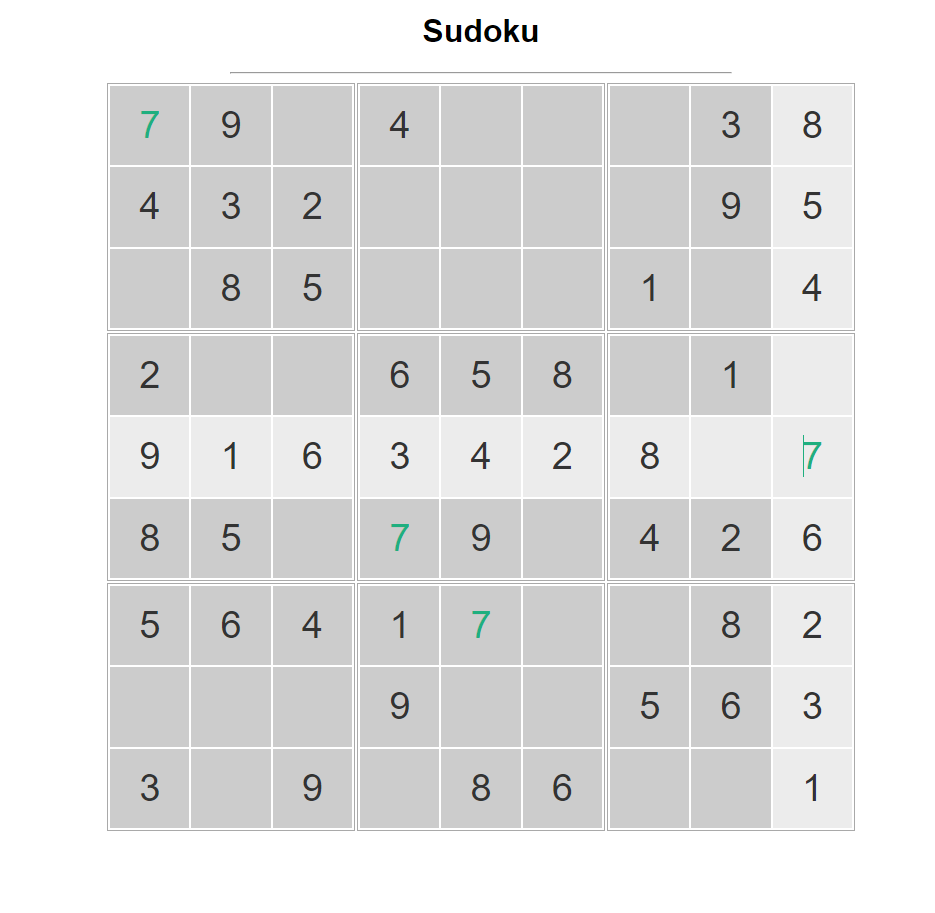
.sudoku-cell.selected-cell

{

    background: #ECECEC;

}

Приклади відображення Судоку:



# ВИСНОВКИ

Під час створення проекту були реалізовані всі поставлені вимоги та цілі:

1. С творення алгоритму генерації ігрового поля
2. Виділення головних секторів, стовпців і рядків
3. Виділення помилок гравця та виділення однакових чисел

Ідеї для покращення проекту

1. Додання таймеру та обмежену кількість похибок для підвищення складності та інтересу гри
2. Додавання різних рівнів складності гри
3. Створення додаткової панелі з числами які будуть допомагати в пошуку чисел
4. Додавання статистики на основі швидкості та точності користувача
5. Створення регістрації для статистики та рейтингу користувачів